ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ №4

Выполнил:
Батуров Максим Дмитриевич
Группа: 6203-010302D

Оглавление

Задание №1

В рамках первого задания я реализовал конструкторы для классов ArrayTabulatedFunction и LinkedListTabulatedFunction, которые принимают массив точек FunctionPoint. Конструкторы выполняют строгие проверки на минимальное количество точек и на упорядоченность абсцисс. Для сравнения вещественных чисел использовался машинный эпсилон. При нарушении условий генерируется исключение IllegalArgumentException.

```java
// конструктор, получающий массив точек
public ArrayTabulatedFunction(FunctionPoint[] points) {  2 usages
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    // проверка упорядоченности точек по X
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i].getX() >= points[i + 1].getX() - EPSILON) {
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
        }
    }

    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 3];

    // создаем копии точек для обеспечения инкапсуляции
    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

```java
public LinkedListTabulatedFunction(FunctionPoint[] points) {  no usages
    this(); // вызов конструктора по умолчанию для инициализации головы

    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    // проверка упорядоченности точек по X
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i].getX() >= points[i + 1].getX() - EPSILON) { // используем EPSILON
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
        }
    }

    // добавляем точки в список, создавая копии для инкапсуляции
    for (FunctionPoint point : points) {
        addNodeToTail(new FunctionPoint(point));
    }
}
```

Задание №2

Был разработан интерфейс Function, определяющий основные операции для функций одной переменной: получение левой и правой границ области определения и вычисление значения в заданной точке. Интерфейс TabulatedFunction был переработан и теперь наследует от Function.

```java
package functions;

public interface Function {
    // возвращает значение левой границы обл. опр. ф-ции
    double getLeftDomainBorder();  11 implementations

    // возвращает значение правой границы обл. опр. ф-ции
    double getRightDomainBorder();  11 implementations

    // возвращает значение ф-ции в заданной точке
    double getFunctionValue(double x);  13 implementations
}
```

Задание №3

В пакете functions.basic созданы классы для аналитически заданных функций. Класс Exp реализует экспоненциальную функцию с использованием Math.exp(). Класс Log вычисляет логарифм по заданному основанию через натуральный логарифм Math.log(). Для тригонометрических функций разработан абстрактный базовый класс TrigonometricFunction с общей областью определения, от которого наследуются классы Sin, Cos и Tan.

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {  2 usages

    // возвращает значение экспоненты в точке x
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }

    // возвращает левую границу
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    // возвращает правую границу
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
}
```

```java
package functions.basic;

import functions.Function;

// класс для триг ф-ций
public abstract class TrigonometricFunction implements Function {  3 usages  3 inhe

    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
}
```

Задание №4

В пакете functions.meta реализованы классы для комбинирования функций. Класс Sum вычисляет сумму двух функций с областью определения как пересечение исходных областей. Класс Mult реализует произведение функций.

Класс Power выполняет возведение функции в заданную степень. Класс Scale обеспечивает масштабирование по осям координат. Класс Shift реализует сдвиг функции вдоль осей. Класс Composition создает композицию двух функций.

Задание №5

Создан служебный класс Functions с приватным конструктором, предотвращающим создание экземпляров. Класс содержит статические методы shift, scale, power, sum, mult и composition, которые возвращают объекты соответствующих функций из пакета functions.meta. Это обеспечивает удобный и безопасный способ создания сложных функций.

```java
package functions;

import functions.meta.*;

public class Functions {  7 usages

    // коснтруктор чтобы нельзя было создать объект класса
    private Functions() {}  no usages

    public static Function shift(Function f, double shiftX, double shiftY) { return new Shift(f, shiftX, shiftY); }

    public static Function scale(Function f, double scaleX, double scaleY) { return new Scale(f, scaleX, scaleY); }

    public static Function power(Function f, double power) { return new Power(f, power); }

    public static Function sum(Function f1, Function f2) { return new Sum(f1, f2); }

    public static Function mult(Function f1, Function f2) { return new Mult(f1, f2); }

    public static Function composition(Function f1, Function f2) { return new Composition(f1, f2); }
}
```

Задание №6

Разработан класс TabulatedFunctions с приватным конструктором, содержащий метод tabulate для создания табулированных аналогов аналитических функций. При нарушении условий генерируется IllegalArgumentException. Метод возвращает ArrayTabulatedFunction с равномерно распределенными точками.

```java
package functions;

import java.io.*;

public class TabulatedFunctions {  11 usages
    private TabulatedFunctions() {}  no usages

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // проверка границ
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
        }

        // проверка минимального количества точек
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        // проверка интервала
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }

        // создаем массив значений ф-ции
        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        // вычисляем значения ф-ции
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            values[i] = function.getFunctionValue(x);
        }

        // возвращаем табулированную ф-цию
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }
```

Задание №7

Реализованы методы для работы с потоками ввода-вывода. Для байтовых потоков используются DataOutputStream и DataInputStream, обеспечивающие эффективную запись примитивных типов. Для текстовых потоков применяются PrintWriter и StreamTokenizer с пробелами в качестве разделителей. Исключения IOException выбрасываются вызывающему коду для гибкой обработки ошибок, при этом потоки не закрываются внутри методов для возможности дальнейшего использования.

```java
// вывод табулированной ф-ции в байтовый поток
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    DataOutputStream dataOut = new DataOutputStream(out);

    // записываем кол-во точек
    dataOut.writeInt(function.getPointsCount());

    // записываем координаты всех точек
    for (int i = 0; i < function.getPointsCount(); i++) {
        dataOut.writeDouble(function.getPointX(i));
        dataOut.writeDouble(function.getPointY(i));
    }

    dataOut.flush();
}

// ввод табулированной функции из байтового потока
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {  1 usage
    DataInputStream dataIn = new DataInputStream(in);

    // читаем кол-во точек
    int pointsCount = dataIn.readInt();

    // читаем координаты точек
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        double x = dataIn.readDouble();
        double y = dataIn.readDouble();
        points[i] = new FunctionPoint(x, y);
    }

    // создаем табулированную ф-цию
    return new ArrayTabulatedFunction(points);
}
```

```java
// запись ф-ции в символьный поток
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    PrintWriter writer = new PrintWriter(out);

    // записываем кол-во точек
    writer.print(function.getPointsCount());
    writer.print(" ");

    for (int i = 0; i < function.getPointsCount(); i++) {
        writer.print(function.getPointX(i));
        writer.print(" ");
        writer.print(function.getPointY(i));
        if (i < function.getPointsCount() - 1) {
            writer.print(" ");
        }
    }

    writer.flush();
}
```

```java
    // считывание ф-ции из символьного потока
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {

        if (in == null) {
            throw new IllegalArgumentException("передан нулевой поток ввода");
        }

        StreamTokenizer tokenizer = new StreamTokenizer(in);
        tokenizer.parseNumbers();

        // считываем число точек
        int tokenType = tokenizer.nextToken();
        if (tokenType != StreamTokenizer.TT_NUMBER) {
            throw new IOException("не найдено количество точек");
        }
        int count = (int) tokenizer.nval;

        // создаем массив для хранения точек
        FunctionPoint[] functionPoints = new FunctionPoint[count];

        // считываем пары
        for (int i = 0; i < count; i++) {
            // считываем x координату
            tokenType = tokenizer.nextToken();
            if (tokenType != StreamTokenizer.TT_NUMBER) {
                throw new IOException("отсутствует x координата для точки " + i);
            }
            double xCoord = tokenizer.nval;

            // считываем y координату
            tokenType = tokenizer.nextToken();
            if (tokenType != StreamTokenizer.TT_NUMBER) {
                throw new IOException("отсутствует y координата для точки " + i);
            }
            double yCoord = tokenizer.nval;

            // создаем точку с полученными координатами
            functionPoints[i] = new FunctionPoint(xCoord, yCoord);
        }

        // возвращаем новую табулированную ф-цию
        return new ArrayTabulatedFunction(functionPoints);
    }
}
```

Задание №8

В классе Main проведено комплексное тестирование функциональности. Протестированы базовые функции Sin и Cos, созданы их табулированные аналоги и проведено сравнение точности. Исследована сумма квадратов табулированных синуса и косинуса. Проверена работа с файлами: экспонента сохранена и загружена из текстового файла, логарифм - из бинарного. Также

понял, что бинарный формат более эффективен по объему, а текстовый более читаем.

Задание №9

Реализована сериализация табулированных функций двумя способами. LinkedListTabulatedFunction использует стандартный механизм Serializable, что удобно, но может приводить к избыточности данных. ArrayTabulatedFunction реализует Externalizable, что дает полный контроль над процессом сериализации. Внешняя сериализация позволяет сохранять только существенные данные - количество точек и их координаты, игнорируя служебные поля и структуры. Это обеспечивает компактность и эффективность, а также независимость от внутренней реализации классов.

```
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable {  7 us
    private FunctionPoint[] points;  37 usages
    private int pointsCount;  37 usages
    private static final double EPSILON = 1e-10;  12 usages
```

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].getX());
        out.writeDouble(points[i].getY());
    }
}

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    int count = in.readInt();
    points = new FunctionPoint[count + 3];
    pointsCount = count;
    for (int i = 0; i < count; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
}
```

```java
public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable {
    private FunctionNode head;  20 usages
    private int pointcount;  22 usages
    private static final double EPSILON = 1e-10;  12 usages

    private static class FunctionNode implements Serializable {  25 usages
        private FunctionPoint point;  3 usages
        private FunctionNode prev;  3 usages
        private FunctionNode next;  3 usages

        public FunctionNode(FunctionPoint point) {  3 usages
            this.point = point; // устанавливаем точку
            this.prev = null; // предыдущий узел пока null
            this.next = null; // следующий узел пока null
        }

        public FunctionPoint getPoint() {  17 usages
            return point; // возвращаем точку
        }

        public void setPoint(FunctionPoint point) {  2 usages
            this.point = point; // устанавливаем новую точку
        }

        public FunctionNode getPrev() {  6 usages
            return prev; // возвращаем предыдущий узел
        }

        public void setPrev(FunctionNode prev) {  6 usages
            this.prev = prev; // устанавливаем предыдущий узел
        }

        public FunctionNode getNext() {  13 usages
            return next; // возвращаем следующий узел
        }

        public void setNext(FunctionNode next) {  6 usages
            this.next = next; // устанавливаем следующий узел
        }
    }
```

```
Сериализация
Функция сериализована в serializable.ser
Функция десериализована из serializable.ser
Сравнение после Serializable:
x=0,0: исходная=NaN, восстановленная=NaN, совпадают=false
x=1,0: исходная=1,000000, восстановленная=1,000000, совпадают=true
x=2,0: исходная=2,000000, восстановленная=2,000000, совпадают=true
x=3,0: исходная=3,000000, восстановленная=3,000000, совпадают=true
x=4,0: исходная=4,000000, восстановленная=4,000000, совпадают=true
x=5,0: исходная=5,000000, восстановленная=5,000000, совпадают=true
x=6,0: исходная=6,000000, восстановленная=6,000000, совпадают=true
x=7,0: исходная=7,000000, восстановленная=7,000000, совпадают=true
x=8,0: исходная=8,000000, восстановленная=8,000000, совпадают=true
x=9,0: исходная=9,000000, восстановленная=9,000000, совпадают=true
x=10,0: исходная=10,000000, восстановленная=10,000000, совпадают=true

Экстернализация
Функция сериализована в externalizable.ser
Функция десериализована из externalizable.ser
Сравнение после Externalizable:
x=0,0: исходная=NaN, восстановленная=NaN, совпадают=false
x=1,0: исходная=1,000000, восстановленная=1,000000, совпадают=true
x=2,0: исходная=2,000000, восстановленная=2,000000, совпадают=true
x=3,0: исходная=3,000000, восстановленная=3,000000, совпадают=true
x=4,0: исходная=4,000000, восстановленная=4,000000, совпадают=true
x=5,0: исходная=5,000000, восстановленная=5,000000, совпадают=true
x=6,0: исходная=6,000000, восстановленная=6,000000, совпадают=true
x=7,0: исходная=7,000000, восстановленная=7,000000, совпадают=true
x=8,0: исходная=8,000000, восстановленная=8,000000, совпадают=true
x=9,0: исходная=9,000000, восстановленная=9,000000, совпадают=true
x=10,0: исходная=10,000000, восстановленная=10,000000, совпадают=true
```